How to Install Literally Anything: A Practical Guide to Singularity

...

Brian DuSell
bdusell1@nd.edu
XSEDE Campus Champions Tech Talk, May 29, 2019


github.com/bdusell/singularity-tutorial


@BrianDuSell

Who Am I?

Brian DuSell

- PhD Student at the University of Notre Dame
- Department of Computer Science and Engineering
- Natural Language Processing Lab
- Email: bdusell1@nd.edu
- GitHub: https://github.com/bdusell
- Twitter: @BrianDuSell
- Not affiliated with Singularity :)

Goals for This Talk

- Segue from Docker to Singularity
- Cover basic Singularity workflow: searching for base images, defining images, and running containers
- Walk through a real example of running a GPU-accelerated Python program with Singularity
- Encourage researchers to release code with container images to foster reproducibility

What is Singularity?

- Singularity is a containerization system like Docker
- Designed with scientific computing in mind rather than cloud applications (more convenient for shared computing environments)
- Allows you to package an execution environment with your code to ensure reproducibility on other machines
- Also allows you to install arbitrary software as a non-root user with minimal effort

Dockerfile

```
FROM ubuntu:18.04
RUN apt-get update -y && \
    DEBIAN_FRONTEND=noninteractive \
    apt-get install -y \
        --no-install-recommends \
        python3 \
        python3-tk \
        python3-pip && \
    rm -rf /var/lib/apt/lists/* && \
    pip3 install torch numpy matplotlib
```

Singularity.def

```
Bootstrap: library
From: ubuntu:18.04

%post
    apt-get update -y
    DEBIAN_FRONTEND=noninteractive \
    apt-get install -y \
        --no-install-recommends \
        python3 \
        python3-tk \
        python3-pip
    rm -rf /var/lib/apt/lists/*
    pip3 install torch numpy matplotlib
```

Why Is It Good?

- Much simpler paradigm than Docker -- container images are just files
  - Makes more sense for scientific computing
- Containers are viewed as one-and-done processes rather than long-running services
- No more Docker daemon or Docker registries
  - Transfer images via `scp`
- Compatible with Docker images!
- Out-of-the-box Nvidia GPU support!

Singularity Solves Two Problems

1. The Portability (or Reproducibility) Problem
   - Writing portable code takes diligence
   - Running other people's code is hard when you are unfamiliar with the frameworks they use
2. The Installation Problem
   - Installing software on an HPC cluster as a non-root user is hard and time-consuming

The Portability Problem

- All code makes assumptions about its environment
- Not always explicitly documented, especially in research
- Code that runs on one machine may fail with cryptic errors on another

Python

```
def f(n):
    return 2 * n
```

Python 3+

```
def f(n):
    return { 2 * x for x in range(n) }
```

Python 3.5+

```
import subprocess

def f():
    return subprocess.run(...)
```

Python 3.5+ and ImageMagick

```
import subprocess

def f():
    return subprocess.run([
        'convert', 'photo.jpg', '-resize', '50%', 'photo.png'])
```

The Installation Problem

- HPC clusters generally give users an account without root access
- Can’t use built-in package managers like yum or apt
- It’s almost always possible to build a library yourself without root privileges, but this can be very time-consuming
- Purposely installing older versions of software to run older code can be tricky
- Software rot

The Solution

What Is a Container?

- Massively popular software isolation technique, particularly thanks to Docker
- An operating system within an operating system, like a VM
- Unlike a VM, it does not virtualize the processor, so there is no overhead for translating machine instructions
- Instead, it shares the kernel of the host OS while spoofing file system and network access via system calls

Terminology

- Container: a process running within a “containerized” environment
- Image: a snapshot of an execution environment from which a container can be instantiated
- Image definition: a deterministic set of instructions for building an image

Why Singularity Solves These Problems

- Portability
  - The image contains all of the program's dependencies save for the OS kernel
  - Runs the same way anywhere Singularity is installed*
- Installation
  - The "root" file system of the image is distinct from the host's
  - Can install software into the image as "root" without affecting the host
  - The resulting image can be run as a container without root privileges

Singularity in a Nutshell

Build an image:

```
$ sudo singularity build example_image.sif example_def.def
```

Download an image:

```
$ singularity pull alpine.sif library://alpine:latest
```

Run a command in a container:

```
$ singularity exec example_image.sif <command>
```

Run with Nvidia GPU support:

```
$ singularity exec --nv example_image.sif <command>
```

Open an interactive shell in a container:

```
$ singularity shell example_image.sif
```

Basic Workflow

1. Define an image in a .def file
    - Choose an appropriate base image
2. Build the image as a .sif file
    - Requires root
3. Instantiate the image as a container
    - Does not require root
4. Tweak image definition until all of your code's dependencies are met

Demo Time

Follow along at:

github.com/bdusell/singularity-tutorial

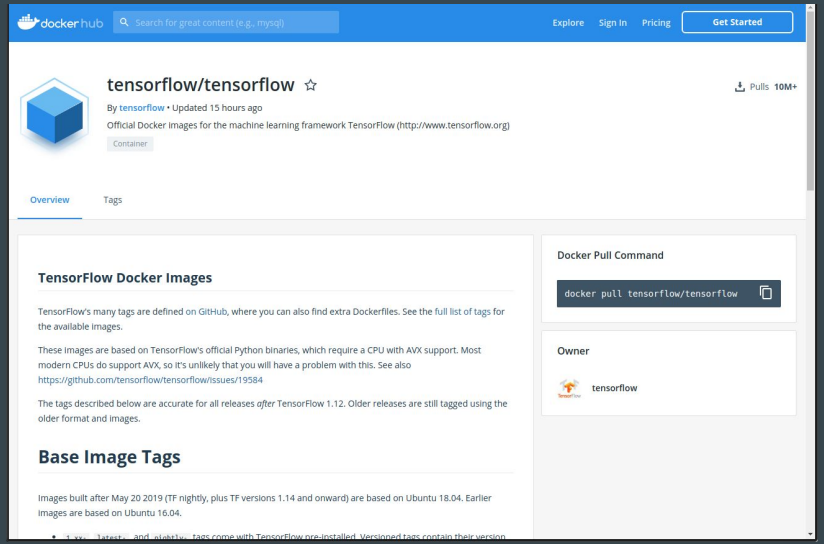
A Simple PyTorch Program

See https://github.com/bdusell/singularity-tutorial/blob/master/examples/xor/train_xor.py

- Trains a neural network to learn the XOR function
- Plots loss as a function of training time using matplotlib
- Saves the trained neural network to a file

Picking a Base Image

- Go-to sources: Docker Hub, Singularity Library, Singularity Hub
- In general, to find an image for library X, just Google “X docker”



Defining an Image

See https://github.com/bdusell/singularity-tutorial#defining-an-image

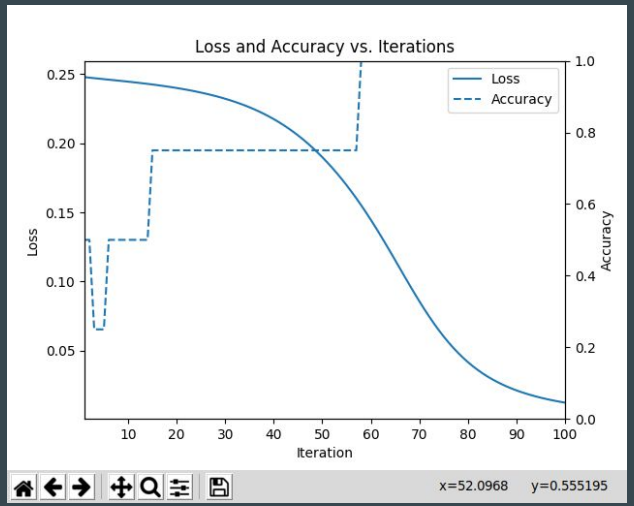
See https://github.com/bdusell/singularity-tutorial/blob/master/examples/xor/version-1.def

Building an Image

See https://bdusell.github.io/singularity-tutorial/casts/version-1.html

Running an Image

See https://asciinema.org/a/Lqq0AsJSwVgFoo1Hr8S7euMe5



Mounting Directories

- Singularity binds several directories between the container and host by default (unlike Docker)
  - Home directory
  - Current working directory
  - Others
- Bind other directories with the `--bind` flag

Environment Variables are Inherited

- Singularity containers inherit environment variables from the host
  - Unless you disable this with `--cleanenv`
- The .def file can contain an `%environment` section for defining environment variables in instances of the image

A Beefier PyTorch Program

See https://github.com/bdusell/singularity-tutorial/tree/master/examples/language-model

- Trains a neural network on Wikipedia text
- GPU-accelerated (requires CUDA)

Picking a Base CUDA Image

See https://hub.docker.com/r/nvidia/cuda

Defining an Image with GPU Support

See https://github.com/bdusell/singularity-tutorial#adding-gpu-support

- Just swap out the base image

Separating Python Modules from the Image

- Right now, installing Python modules requires re-building the image
- But Python modules can easily be installed without root privileges!
- Let's use a package manager for Python
  - Installs packages in the current working directory on the host, but runs them in the container
- Only rely on the image for the basic Ubuntu/CUDA/Python environment (things that require root or change infrequently)
- Makes the image smaller and more reusable

Using Pipenv

- Pipenv is a package manager for Python
  - `pipenv install <module>`
- Based on virtualenv / venv
- Stores packages in a local directory named `.venv/`
- Tracks Python dependencies in version-controlled files named `Pipfile` and `Pipfile.lock`
- Need to run `pipenv install` once in the container to initialize `.venv/`
- Run Python programs with:

```
$ singularity exec image.sif pipenv run python example.py
```

See https://github.com/bdusell/singularity-tutorial#separating-python-modules-from-the-image

The Container-y Way to Write Portable Code

- Make an image definition part of your code base, and continually rebuild your image from it
- Install whatever dependencies you need (e.g. Python) in the image
- Develop your code as usual, but every time you run it, run it inside Singularity using your image
- Add dependencies to your image definition as needed
- Doing this forces you to build up your environment from scratch...
- ...but in the end it will work everywhere

Crafting Reproducible Code

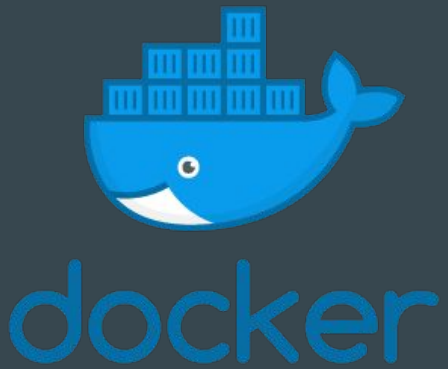
- Requires two things
  - Develop within the confines of a widely available execution environment
    - e.g. Singularity, Docker, Python, Node.js
    - Developing within containers ensures you do not accidentally violate those boundaries
  - Provide an automated way of setting up the execution environment on another host
    - sudo singularity build / singularity pull
    - pipenv install + Pipfile
    - npm install + packages.json
    - spack install + spack.yaml

A New Best Practice

- Develop your projects in containers!
- Release the definition file with your project
- You can even upload a pre-built image to the official Singularity Library

Singularity vs. Docker

- Docker is designed primarily for cloud services, with multiple containerized services communicating among one another (e.g. a web server and a database)
- Docker has a centralized daemon that manages all containers and images running on the host; Singularity does not
- Singularity has no image layers
  - Rebuilds can take a long time
- Singularity just added support for multi-stage builds



Docker as a Development Tool

- Docker is well worth exploring too!
- A tool I made for streamlining my workflow with Docker containers:

  github.com/bdusell/dockerdev

Singularity vs. Kubernetes

- Kubernetes lets containers team up together
- Singularity does not ship with tools for managing multiple containerized services
  - Because we're not building web servers
- Singularity can sit inside Kubernetes

kubernetes

Singularity vs. Spack

- Spack solves the same problems without using containerization
  - Reproducibility via spack install + spack.yaml
- Spack is also a powerful build tool which can facilitate hardware-specific optimizations
- Spack can happily live inside of Singularity
- With Singularity, you don’t need to build anything, you just need to download it

Thank You!

https://github.com/bdusell/singularity-tutorial